# Parallel Application Placement onto 3-D Reconfigurable Architectures

Panayiotis Danassis*, Kostas Siozios† and Dimitrios Soudris‡
School of Electrical and Computer Engineering, National Technical University of Athens, Greece
Email: *panosd@microlab.ntua.gr, †ksiop@microlab.ntua.gr, ‡dsoudris@microlab.ntua.gr

***Abstract*—Placement is considered one of the most arduous and time-consuming processes in physical implementation flows for reconfigurable architectures, while it highly affects the quality of derived application implementation as it is tightly firmed to the total wirelength and hence the maximum operating frequency. This problem becomes more acute for three-dimensional (3-D) architectures since the complexity of such architectures imposes additional challenges that have to be sufficiently addressed. Throughout this paper we introduce a novel placement algorithm, targeting 3-D reconfigurable architectures, based on Ant Colony Optimization (ACO). Experimental results validate the effectiveness of our algorithm since it achieves 10% reduction in the critical path delay on average. Additionally, in contrast to relevant approaches which are executed sequentially, the proposed algorithm exhibits inherent parallelism and can take full advantage of today's multi-core architectures.**

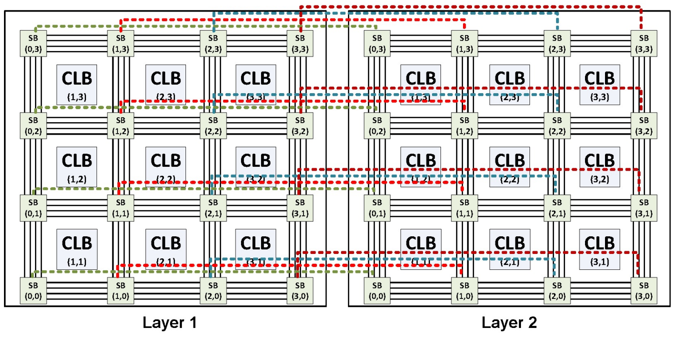

Fig. 1: Architectural template of the proposed 3-D FPGA.

## I. Introduction

Field-Programmable Gate Arrays (FPGAs) have become the implementation medium for the majority of digital circuits. For decades, semiconductor manufacturers have been shrinking the size of the transistors to achieve the yearly increase in performance, as described by Moore's Law, which exists only because the RC delay used to be negligible as compared to the signal propagation delay. For sub-micron technology, however, that is not the case. Furthermore, previous studies have shown that at 130nm technology node, approximately 51% of the microprocessor's power is consumed by the interconnect fabric [1]. This has generated many discussions concerning the end of device scaling as we know it, and has hastened the search for solutions beyond the perceived limits of the current 2-D devices.

Three dimensional (3-D) chip stacking is considered by many as the silver bullet technology that will accommodate the aforementioned shortcomings [2]. Stacking multiple dies in the vertical axis and interconnecting them using very fine-pitch Through Silicon Vias (TSVs) enables the creation of chips with shorter interconnect, on average, which in turn leads to reduced signal propagation delay [2], [3]. Additionally, stacking smaller dies rather than manufacturing a large planar yields significant cost improvements.

The benefits of using 3-D integration are especially great for designing FPGAs since these architectures suffer from data communication problems; interconnection delay and power consumption are the main bottlenecks in such architectures. However, in order for such technologies to be widely accepted, several challenges have to be satisfied. As a result, there is an ever growing need for more efficient CAD tools that support application mapping onto 3-D platforms and are able to produce results in a reasonable execution run-time. Throughout this paper we introduce a novel algorithm, based on Ant Colony Optimization (ACO) [4], for addressing the placement problem on such architectures.

The rest of the paper is organized as follows: Section **??** describes the architecture of the underlying 3-D FPGA, Section III presents the proposed algorithm and Section IV provides a number of qualitative results to prove its efficiency. Finally Section V concludes the paper.

## II. Architectural Template of the Targeted 3-D FPGA

The proposed architectural template that targets to alleviate the impact of long wire-lengths is depicted in Figure 1. Note that the architectural template discussed in this paper is orthogonal to the rest of the approaches for 3-D FPGAs. Each layer of the FPGA is based on an island-style architecture. The communication between resources assigned to different layers is provided by vertically aligned TSVs. These TSVs are implemented inside the Switch Boxes (SBs), which are appropriately extended in order to be aware of the third dimension [5]. This kind of connectivity provides routing paths (depicted in dotted lines) between SBs assigned to adjacent layers with the same $(x, y)$ coordinates.

Regarding the modeling of the remaining hardware resources (e.g. TSVs, routing wires, transistors, etc), we follow a similar approach to the one found in relevant literature [6]. Note that the selection of the employed values for the architectural components do not affect the generality of the introduced solution, which is applicable to other flavors of 3-D integration as well (such as the 2.5-D provided by Xilinx).

## III. 3-D FPGA Placer based on ACO

In this section we introduce the proposed algorithm for addressing the placement problem on 3-D reconfigurable architectures. Our approach relies on Ant Colony Optimization (ACO). It is a novel, swarm-intelligence based algorithm, that mimics the foraging behavior of certain species of ants in order to find a high quality placement in regard to legality constrains and optimization goals. The inherent parallelism of the ACO algorithms, the flexibility they provide in regard to integrating different cost functions or FPGA architectures and the fact that they combine a positive feedback mechanism and a stochastic decision policy which account for rapid discovery of good solutions are just some of the strengths that make ACO algorithms a good fit for the problem of FPGA placement. The pseudo-code for our ACO-based placer is presented in Algorithm 1.

**Algorithm 1** Pseudo-code of our ACO-based placer.

```
 1: while (!termination_condition()) {
 2:   for (n = 1; n ≤ n_ants; n++) {
 3:     construct_solution(ant[n]);
 4:     compute_placement_quality(ant[n]);
 5:     compare_best_so_far_ant(ant[n]);
 6:     local_pheromone_update(ant[n]);
 7:   }
 8:   global_pheromone_update();
 9:   iteration++;
10: }
```

Several ACO algorithms have been proposed in the literature [4]. Our implementation incorporates concepts from the $MAX - MIN$ Ant System (MMAS) and the Ant Colony System (ACS). The functionality of our introduced algorithm can be summarized as follows: In every iteration, each ant in the colony constructs a solution from scratch. To do so, every $ant[n]$ assigns block $i$ to a physical location $j$ on layer $k$ using a pseudorandom proportional rule (Eq. 1) which utilizes pheromone trails and heuristic information. As a result, all the blocks in the circuit netlist will be mapped to a specific location on the FPGA. Succeeding the construction of a solution, each ant evaluates the quality of its solution and based on that quality it decides on how much pheromone to deposit on the utilized components or connections. The update of the pheromone trails consist of both evaporation and new pheromone deposition. Subsequent ants use this information to guide their search towards promising solutions. Thus, good-quality solutions emerge as the result of the collective interaction among the ants. The process continues until one of the termination conditions is met. Summarizing:

- A colony of artificial ants moves *concurrently*, *asynchronously* and *independently* on the construction graph, incrementally building solutions for the placement problem.
- In order to move around the graph they use a *stochastic* local decision policy that makes use of the *pheromone trails* and the *heuristic information*.
- Based on the quality of the solution, each ant *deposits* different amounts of pheromone. In doing so they adaptively modify the way the problem is represented and perceived by other ants.

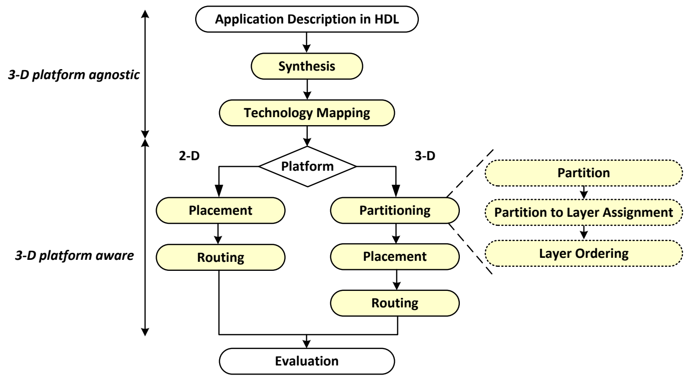

Fig. 2: Toolflow for application mapping onto 3-D FPGAs.

In the following subsections we provide additional technical details regarding the different aspects of the employed ACO algorithm.

### A. Heuristic Information & Pheromone Trails

The heuristic information is used to guide the ants in the early stages of the algorithm, before the collected pheromone starts to take effect. Even though convergence is guaranteed [4], the time to convergence is uncertain. That is why we need a good heuristic metric to speedup the process during the initial construction steps. In this implementation we introduce two types of heuristic information: a *static* one, which is a priori defined and tends to place more centric blocks that connect to many other blocks [7] and a *dynamic* one which is more aggressive and tries to minimize the Manhattan distance of blocks of the same net.

After some iterations, the collective knowledge of the ants is incorporated into the pheromone trails. The initialization of those trails plays a determinant role in the performance of the algorithm and the speed of convergence. The formula we used to initialize them is $\tau_0 = 1/\rho C^*$, where $C^*$ is the cost of the best placement we forecast (approximately 40% of the initial cost) and $\rho$ the evaporation rate, since we estimate that in the long run, the upper pheromone trail limit on any component is bounded by $1/\rho C^*$.

For each possible assignment of a netlist block $i$ to a physical location $j$ on a layer $k$ we have a heuristic value $\eta_{ijk}$ and a pheromone value $\tau_{ijk}$.

### B. Solution Construction

In Figure 2 we present the design process for application mapping onto 3-D Reconfigurable Architectures. One of the key advantages of our implementation is the ability to incorporate in the solution construction phase both the stages of partitioning and placement and as a result has the potential to arrive at superior placement solutions. In the solution construction phase we use the pseudorandom proportional rule from ACS. In particular, the probability with which ant $m$ places the netlist block $i$ to the physical spot $j$ on the layer $k$ is given by Equation 1:

$$p_{ijk}^{m} = \frac{[\tau_{ijk}]^{\alpha} \times [\eta_{ijk}]^{\beta}}{\sum\limits_{(y,z)\in N^{m}} [\tau_{iyz}]^{\alpha} \times [\eta_{iyz}]^{\beta}} \tag{1}$$

where $\alpha$, $\beta$ are parameters, and $N^m$ is the set of available positions that ant $m$ has in its disposition. To speedup the process, Equation 1 is used to map only a portion of the blocks. Specifically we use Equation 1 with probability $(1-q_0)$, were $q_0$ is a parameter, while with probability $q_0$ we use Equation 2 which maps block $i$ to the best location found so far $J$ on layer $K$. As a result with probability $q_0$ the ant makes the best possible move as indicated by the learned pheromone ($\tau_{ijk}$) and heuristic ($\eta_{ijk}$) information (exploitation), while with probability $(1 - q_0)$ it performs a biased exploration. Tuning the parameter $q_0$ allows modulation of the degree of exploration and the aggressiveness of the system.

$$(J,K) = \underset{(y,z)\in N^{m}}{\operatorname{argmax}} \left\{ [\tau_{iyz}]^{\alpha} \times [\eta_{iyz}]^{\beta} \right\} \tag{2}$$

This random proportional rule is analogous to the "Roulette Wheel" selection method [8] used in genetic algorithms in the sense that fittest individuals (the ones who in the past have produced hight quality solutions) have a larger share of the roulette wheel (higher pheromone values) and as a result greater probability to be chosen again, where weakest individuals occupy smaller share of the roulette wheel (lower pheromone values) and have smaller probability to be chosen.

## C. Cost Function

After every block of the netlist has been successfully placed by an ant, it is time to evaluate the placement's quality. Here the optimization goals that we used were the minimization of the total wire-length ($wiringCost$) and the minimization of the delay of the circuit ($TimingCost$). The cost function is presented in Equation 3 and can be found in more detail in [6]. It is widely accepted and it is proved to be an effective way to evaluate a placement's quality.

$$Cost = \lambda \times timingCost + (1-\lambda) \times wiringCost \tag{3}$$

## D. Pheromone Update

Finally, we have the process of the pheromone update. The aim of this stage is to reinforce the trails associated with good solutions while, using the evaporation mechanism, decrease the ones that produced poor solutions. We have two stages of pheromone update: a local and a global one.

The global pheromone update stage is performed at the end of every iteration and implements the evaporation and the new pheromone deposition process using Equation 4. On the other hand the local pheromone rule is applied after an ant constructs a complete placement to discourage subsequent ants of the same iteration from constructing a similar placement, effectively increasing the exploration space.

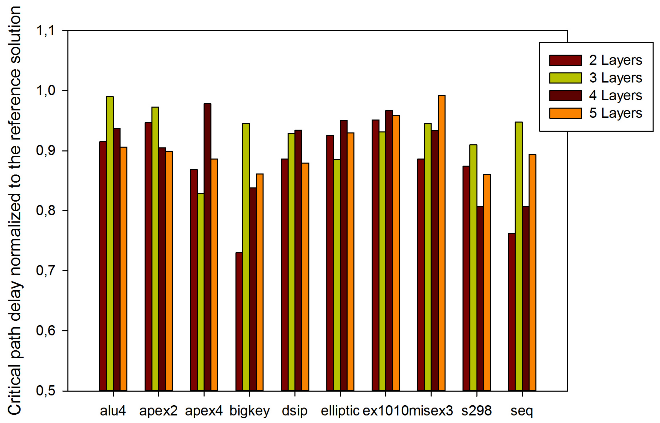

Fig. 3: Critical Path Delay.

$$\tau'_{ijk} = \begin{cases} \tau_{min} & \text{if } \tau'_{ijk} < \tau_{min} \\ (1-\rho) \times \tau_{ijk} + \frac{1}{Cost^{best}} & \text{if } ant^{best} \text{ maps } i \rightarrow (j,k) \\ (1-\rho) \times \tau_{ijk} & \text{otherwise} \\ \tau_{max} & \text{if } \tau'_{ijk} > \tau_{max} \end{cases} \tag{4}$$

## E. Parallel Implementation

Since ants move concurrently and asynchronously, ACO algorithms are inherently parallelizable both in the data and population domains [4]. Although the use of the local update rule of the ACS can lead to communication overhead, we can overcome this by implementing a coarse-grained approach with rather rare information exchange. In other words, for the parallel implementation we disregarded the local pheromone update rule and let all the ants move in parallel, exchanging information only at the end of every iteration. Due to the use of a much larger colony in parallel implementations, there were no considerable disadvantages from the omission of the local update rule, since the much larger number of ants can effectively search a much greater solution space. That led to a very simple implementation with considerable benefits in runtime.

## IV. Experimental Results

## A. Experimental Setup

The targeted 3-D FPGA consists of two to five layers, with identical logic and routing resources among these layers and six TSVs per 3-D SB. The introduced ACO-based placer was integrated as part of the open-source toolflow 3-D MEANDER [5]. The measurements were taken using a set of 10 MCNC benchmarks. For the reference solution we employed the TPR tool [9]. Note that both the above tools perform netlist routing using the negotiated pathfinder algorithm. Consequently, we expect that performance improvement is based only on the different placement algorithms. The experiments were performed on an Intel Core 2 Quad CPU Q9400 clocked at 2.66GHz. Four different colonies were utilized that concurrently searched for a good placement solution. In the end the best one was chosen.

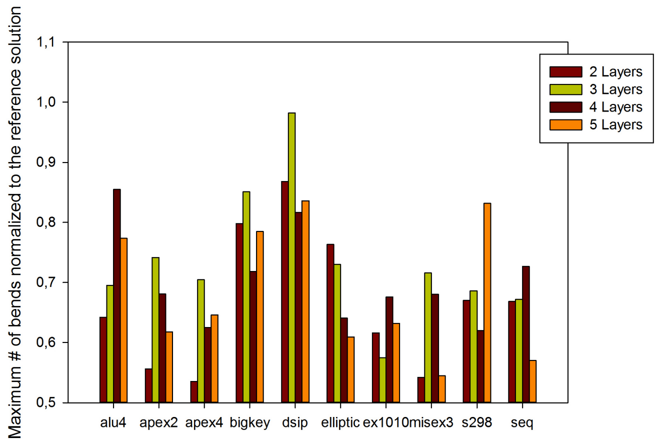

Fig. 4: Number of bends.

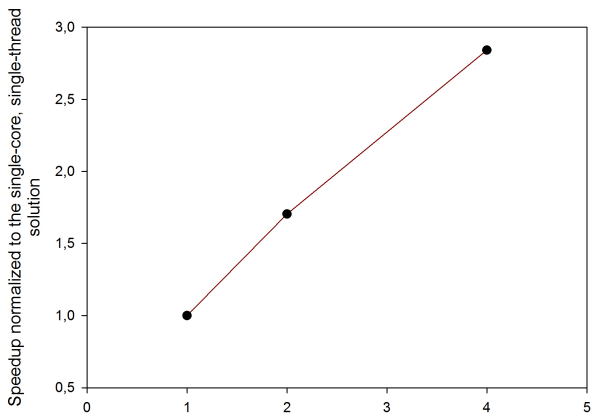

Fig. 6: Multi-core speedup.

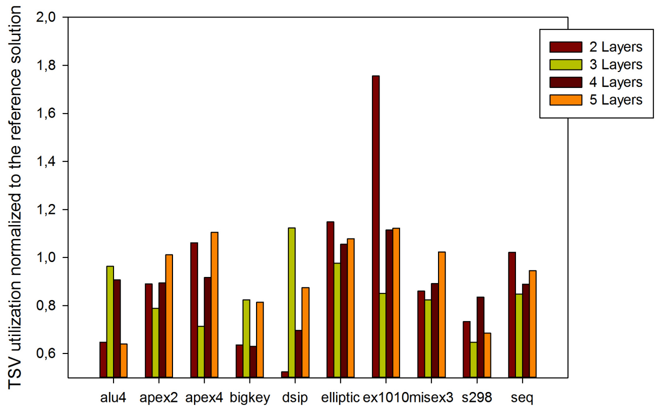

Fig. 5: TSV utilization.

## B. Experimental Results

Our algorithm achieves 32% reduction of the maximum net length and 29% reduction of the maximum segments used by a net, on average. As a result we achieve 10% reduction of the critical path delay, on average, as depicted in Figure 3 which gives the critical path delay normalized to the reference solution. Hence, besides the increase of the maximum operating frequency, we expect to have an improvement in power consumption. At the same time we achieve 30% improvement in the maximum number of bends, as depicted in Figure 4. That leads to fewer transistors inside the switch boxes (SBs) and as a result lower fabrication cost. Furthermore our algorithm achieves a better manipulation of the available resources, compared to existing tools, since it presents 10% lower TSV utilization, on average, even though it achieves smaller critical path delay. The TSV utilization is presented in Figure 5.

By taking advantage of the inherent parallelism of our algorithm and the existing multi-core platforms we can considerably reduce the execution run-time. Figure 6 depicts the on average speedup achieved in a 2-core and a 4-core processor with identical clock frequency. The results are normalized over the corresponding single-core and single-thread execution. As illustrated, our proposed algorithm achieves almost a 3× speedup on a 4-core CPU, allowing it to take take full advantage of today's multi-core architectures. Hence by utilizing the inherent parallelism we can either keep the run-time in check, produce higher quality results in the same run-time or a combination of both, depending on our needs.

## V. Conclusion

In this paper we introduced a novel 3-D FPGA placer. As mentioned, our algorithm achieves 10% reduction on the critical path delay, on average. Moreover by utilizing it's inherent parallelism, we can take full advantage of today's multi-core architectures, further decreasing the execution run-time as multi-core CPUs scale according to today's market trends.

## Acknowledgments

This work has been partially supported by the TEACHER project which is funded by the DAAD (2014).

## References

[1] N. Magen, A. Kolodny, U. C. Weiser, and N. Shamir, "Interconnect-power dissipation in a microprocessor." in *SLIP*, L. Scheffer and I. L. Markov, Eds. ACM, 2004, pp. 7–13.

[2] V. F. Pavlidis and E. G. Friedman, *Three-dimensional Integrated Circuit Design*. USA: Morgan Kaufmann Publishers Inc., 2009.

[3] A. Papanikolaou, D. Soudris, and R. Radojcic, *Three Dimensional System Integration: IC Stacking Process and Design*, ser. SpringerLink : Bücher. Springer, 2010.

[4] M. Dorigo and T. Stützle, *Ant Colony Optimization*. USA: Bradford Company, 2004.

[5] K. Siozios, V. F. Pavlidis, and D. Soudris, "A novel framework for exploring 3-d fpgas with heterogeneous interconnect fabric," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 5, no. 1, pp. 4:1–4:23, Mar. 2012.

[6] V. Betz, J. Rose, and A. Marquardt, Eds., *Architecture and CAD for Deep-Submicron FPGAs*. USA: Kluwer Academic Publishers, 1999.

[7] W. Xu, K. Xu, and X. Xu, "A novel placement algorithm for symmetrical fpga," in *7th Int. Conf. on*, Oct 2007, pp. 1281–1284.

[8] M. Mitchell, *An Introduction to Genetic Algorithms*. USA: MIT Press, 1998.

[9] C. Ababei, "Tpr: Three-d place and route for fpgas." in *FPL*, ser. Lecture Notes in Computer Science, J. Becker, M. Platzner, and S. Vernalde, Eds., vol. 3203. Springer, 2004, p. 1172. [Online]. Available: http://dblp.uni-trier.de/db/conf/fpl/fpl2004.html#Ababei04